# ABERSOFT FORTH

John Jones-Steele

48K SPECTRUM

M

# CHAPTER 1. INTRODUCTION

## 1.0 Starting Out.

This manual is not intended to be a tutorial of FORTH, there are many books available on the market that do this more than adequately, however all additions to the standard are described with examples, so that they may be used in your own definitions. For the beginner to the Language, a highly recommended book is 'Starting Forth' by Leo Brodie.

For the more advanced user one useful reference work is 'The Systems guide to fig-Forth' by C. H. Ting.

FORTH is a language which combines the features of high level languages with the speed of machine code.

The FORTH interpreter interprets each line of commands by searching through an internal dictionary of words which it understands. It then acts on these words. Some of the words allow creation of new words, which in turn form the basis for even high level words, Eventually a whole program comes together, and is called by a single word. This gradual development allows for better checking of sections of a program than BASIC does.

## 1.1 Setting Up.

To install the basic system on your Spectrum type either

```
    LOAD ""
or
    LOAD "FORTH"
```

When the compiler has loaded, you will be greeted by the message:-

48k SPECTRUM fig-FORTH n.nn (where n.nn is the version number) Abersoft: 1983

and a flashing C as a cursor. All commands may be entered in upper case or lower case depending on how the words were defined. (i.e. TEST, Test and test are all different words and are treated as such by the compiler.) Upper and lower case are toggled in the normal Spectrum manner as in the Graphics mode. The cursor changes to a C, L or G respectively. FORTH does not use the key-word system of Sinclair BASIC and all commands must be input in their entirety. All the keyboard symbols are available by using just the symbol shift (i.e. '[' which normally needs extended mode Y just requires symbol shift Y). The only character not available by this method is the copyright symbol.

There are now two Break keys available in this system, the standard key which can be dangerous when doing cassette or printer I/O, or the Caps Shift 1 which can be safely trapped by ?TERMINAL.

To test that the system is running, just hit the ENTER key and the system should respond

```
ok
```

You are now ready to begin programming in FORTH.

# CHAPTER 2. USING SPECTRUM FORTH

## 2.0 The Stack

One of the major differences between FORTH and most other high-level languages is that FORTH uses Reverse Polish Notation (RPN). Normally, the operator '+' comes between the numbers you wish to add (e.g. 4 + 7). In RPN, the operator comes after the numbers (i.e. 4 7 + instead. Note that you need to insert spaces between the 4, 7, and +). This is because a stack is used to store numbers when evaluating expressions. (In case you don't know what a stack is, imagine a file of plates. The last plate put on the pile, being on top, is the first to come off again). When FORTH finds a number, it puts it onto the stack. When it finds an operator, it takes the required numbers off the stack, then puts the result back onto the stack. The word dot '.' (without the quotes) takes a number off the top of the stack and prints it.

The BASIC program line 'PRINT (3 + 7) * (8 + 2)' is, in FORTH, '3 7 + 8 2 + * .' (note the spaces are important).

| After executing: | the stack becomes: |
|---|---|
| 3 | (3) ← top of stack on this side |
| 7 | (3, 7) |
| + | (10) |
| 8 | (10, 8) |
| 2 | (10, 8, 2) |
| + | (10, 10) |
| * | (100) |
| . | () 100 is printed |

Practice is the only way to get familiar with RPN. Similar words explained in the Glossaries are:
+ – / */ */MOD /MOD MOD MAX MIN AND OR XOR MINUS +– 1+ 2+

There are some words which shuffle the numbers on the stack
DUP duplicates the top number
DROP discards the top number
SWAP swaps the top two numbers

So, for example:
1 DUP . . prints 1 1 ok
1 2 DROP . prints 1 ok
1 2 SWAP . . prints 1 2 ok
Similar words: ROT OVER
See also: S0 SP@

## 2.1 Variables and Constants

A variable in FORTH must be explicitly created before it is used, using the word 'VARIABLE'.

3 VARIABLE A1
will create a variable 'A1' with initial value 3. Any sequence of non-blank characters will work as a name.
6 A1 !
stores 6 in A1 once A1 has been defined as a variable.
A1 @ .(can also be written as A! ?)
will fetch the contents of 'A1' and print it. The word 'A1' actually puts the address of 'A1' on the stack. '!' takes an address off the stack then takes a number off the stack and stores it at the address. The work '@' takes an address off the stack and replaces it with the contents of memory at that address. The word '?' is the equivalent of '@ .' for those who are lazy. See also the words '+!'.

A constant in FORTH is a fixed number which is given a name, using the word, 'CONSTANT'.
10 CONSTANT TEN
creates a constant, 'TEN', with value 10. From now on each time the work 'TEN' is found, 10 will be pushed onto the stack, so
TEN .
will print:
10 ok
Note that '!' and '@' are not needed with constants.

Among the predefined system variables there is 'BASE'. 'BASE' contains the current number base used for input and output. 'HEX' stores $16^{10}$ in 'BASE', setting hexadecimal mode. 'DECIMAL' stores $10^{10}$ in 'BASE', setting decimal mode.

## 2.2 Types of Numbers

FORTH is a typeless language. There is no distinction between a number and a character for example. Things on the stack are taken by a word and interpreted as that word sees fit. Some of the more common interpretions, other than the normal 16 bit signed numbers (range −32768 to 32767) which have been used, are:
Unsigned:
range 0 to 65535. The word 'U.' acts like '.' but it assumes that the number is unsigned rather than signed.
Double precision:
The top two numbers on the stack are interpreted as a 32 bit number, either signed or unsigned. The top number is taken as the high 16 bits. The second number is taken as the low 16 bits. To put a 32 bit number onto the stack type the number with a decimal point somewhere in the number. The system variable DPL contains the number of digits following the decimal point, in case it is important.
70.000 . . DPL @ .
will print 1 4464 3 ok. The 32 bit number 70000 (not 70) is put onto the stack as two numbers. The first '.' prints the top half as a signed number. The bottom half is then printed (note 7000 = $1 \times 2^{16}$ + 4464). The contents of DPL are 3 since there are 3 digits after the decimal point.

The words U* U/MOD M* M/ and M/MOD are mixed mode versions of * / and /MOD where the operands and results are of different types.

Byte:
Sometimes only 8 bit numbers are required, for instance to represent a character. They are represented on the stack by 16 bit numbers with the 8 highest bits zero. 'C@' and 'C!' fetch and store only a single byte at a time.
Flags:
To make decisions, the values true and false are needed. For instance

```
2 3 = .
```

prints 0 (false), whereas

```
3 3 = .
```

prints 1(true). In fact any non-zero number is treated as true. (So, '−' can be used for <>, since x − y = 0 (false) only if x = y.) Other comparisons are:
< U< 0< (equivalent to 0 <) and 0= (equivalent to 0 =).

'NOT' changes a true flag to false and vice versa. (This is actually equivalent to 0=.)

## 2.3 FORTH modes

FORTH has two different ways of working. One is the interpretive mode and the other is the defining (or compiling mode).

In the interpretive mode, FORTH takes whatever is input and tries to execute it immediately. If you type

4 7 + . (dot being the FORTH word for print)

the computer will give an immediate answer of

```
11 ok
```

However in compiling mode, FORTH takes what is input and stores it away in its dictionary and uses it later, As an example, if you wanted to input the sum above, but only see the result later (a rather strange thing to want to do) you could define a new word, thus

```
: STRANGE 4 7 + . ;
```

(: is the word used by FORTH to mean begine compiling, ; is the word for finish compiling). If you then type

```
STRANGE
```

and 'enter', the computer will then give you

```
11 ok
```

Once a word is defined in this way, it can be used in other definitions

```
: TOOSTRANGE STRANGE STRANGE ; TOOSTRANGE
```

compiles a word 'TOOSTRANGE' which will perform strange twice, then executes the word, printing

```
11 11 ok
```

## 2.4 Control Structures

The FORTH structure IF ... ENDIF (or IF ... ELSE ... ENDIF) allows conditional execution (only) within a definition. The equivalent of BASIC's 'IF A < 2 THEN PRINT " TOO BIG" ' is

```
:TEST  A@  2<IF. "  TOO  BIG"  ENDIF
```

'A @ 2 <' puts a flag on the stack stating whether A < 2. IF removes the flag. If the flag is true (non-zero) then the following code is executed. The word ' ." ' prints everything up to the next double quote ' " '. (Note that there has to be a space after ' ." ' for it to be recognised properly). If the flag is false (zero) then the code up to ENDIF is skipped. In this case of IF ... ELSE ... ENDIF, if the flag is true the code between IF and ELSE is executed and the code between ELSE and ENDIF is skipped. If the flag is false, only the ELSE ... ENDIF code (and that which follows ENDIF as usual) is executed. Inside any IF ... ENDIF clause you can have another one.

BASIC's 'FOR A = 1 TO 10 : PRINT A : NEXT A' is FORTH's

```
: TEST 11 1 DO I . LOOP ;
```

When 'TEST' is exucuted, 'DO' takes two numbers off the stack. The top number (1 here) is the starting value. The second number (11) is the limit. I copies the loop index (A in the BASIC program) onto the stack, where it is printed by '.'. 'LOOP' increments the loop index by 1. If the limit is *reached* or exceded, execution continues as normal, otherwise it loops back to the 'DO'. Hence the limit being 11 in order to count to 10. Note that the loop is done at least once no matter what the limit is. 'n +LOOP' instead of 'LOOP' is FORTH's equivalent of 'STEP n' in BASIC. If loops are nested, 'J' returns the index of the outer loop. ' I' ' returns the limit of the inner loop. 'LEAVE' changes the limit so that the loop will be left as soon as 'LOOP' is reached.

Another type of loop is BEGIN ... flag UNTIL.

```
: TEST BEGIN   . . .   A @ 2 > UNTIL ;
```

is the equivalent of

```
10 REM BEGIN
20 . . .
30 IF NOT (A > 2) THEN GOTO 10
```

i.e. . . . will be executed over and over again until A > 2.
See also: BEGIN ... AGAIN and BEGIN ... flag WHILE ... REPEAT

One of the most useful commands in a language such as Pascal, is the CASE statement. This allows for the testing of a number for many different values and executing different procedures on each value. This is available in standard FORTH only by using many nested IF ... IF ... IF ... ENDIF ENDIF ENDIF and can be very tedious. A new structure in Spectrum FORTH is naturally called the CASE structure. Its use is:-

```
...(instructions leaving a single value on the stack)
CASE
  8  OF ."This is 8" CR ENDOF
```

```
12  OF ." This is 12" CR ENDOF
99  OF ." This is 99" CR ENDOF
ENDCASE
```

This structure provides a much more readable and less error prone way of making multiple decisions. In the above example, if 99 was left on the stack : This is 99 : would be printed. Any value other than 8, 12 or 99 would produce no output at all.

## 2.5 Keyboard and Screen I/O

You have met the word ' ." ' which prints a fixed message onto the screen. 'EMIT' expects the ASCII code for a character on the stack. It then prints that character.

```
  65 EMIT
```

will print the letter 'A' (ASCII code 65). 'TYPE' is used for printing strings. The variable 'TIB' contains the address of the Terminal Input Buffer where what you type is stored. To type out the first 10 characters in the buffer, you need to supply 'TYPE' with the starting address, and the number of characters to be printed:

```
  TIB @ 10 TYPE
```

will print

```
  TIB @ 10 T
```

'SPACES' takes a number, n from the stack and prints out n spaces.

'KEY' waits for a key to be pressed, then puts the ASCII code of that key onto the stack. 'EXPECT' requires an address and a count, just like 'TYPE'. However, 'EXPECT' takes the specified number of characters from the keyboard (or all the characters up to 'enter') and tacks one or two nulls (ASCII 0) on the end. The word 'QUERY' is defined as 'TIB @ 80 EXPECT'.

```
  INKEY
```

will return the ASCII code of the key currently being pressed. If no key is being pressed, the value 255 is returned. This word is not standard FORTH, but is included for Spectrum FORTH users.

## 2.6 Vocabulary

'VLIST' prints out all known words in the current vocabulary. (A vocabulary is a subsection of the whole dictionary. The normal vocabulary is called FORTH and is selected using the word 'FORTH'.)

```
  FORGET word
```

will forget all words defined from 'word' onwards. A handy thing to do is to compile the word 'TASK' as the first new word

```
  :TASK ;
```

Then, if after compiling more words, you decide to get rid of them all,

```
  FORGET TASK
```

will do the job. The system variable 'FENCE' contains an address, below which you cannot FORGET a word. To protect the words you have just compiled type 'HERE FENCE !'

To create a new vocabulary 'MYWORDS
type

VOCABULARY MYWORDS IMMEDIATE

The word 'MYWORDS' will now cause this new vocabulary to be *searched* when *interpreting* words (the system variable 'CONTEXT' is set to MYWORDS). *New definitions* will, however, be *added* to the old vocabulary (the system variable 'CURRENT' is still pointing to FORTH). To select the new vocabulary as the one to add new definitions to, type:

MYWORDS DEFINITIONS

This sets the current vocabulary to the context vocabulary (made MYWORDS by 'MYWORDS'). To go back to adding definitions to the FORTH vocabulary, type

FORTH DEFINITIONS

Some words, such as 'FORTH', are immediate. This means that they will be executed, even in compile mode. These words can be distinguished in the glossary, by the letter P at the top of the definition.

## 2.7 The RAM disc

One of the problems with using FORTH on a non-disc system is that once a word has been defined the original source for that definition is lost. For a large definition, this is an obvious nuisance if it is found that it does not subsequently do what was expected of it. In Spectrum FORTH this has been got round by using the top of store as a pretend, small disc of 11k bytes in total size. This may seem like a small amount, but given FORTH's compactness, a surprisingly large application can be written. As an example, the editor described later, took up only 8 pages of this disc (a page is 1024 bytes long arranged as 16 lines of 64 characters, a strange arrangement for the Spectrum display, but this is what is required by the FORTH standard) and that included copious comments. This also means that when the upgrade to MICRO-DRIVES is available, the changes to your way of working should be almost unnoticeable, except for the seeming vast increase in store!

The pages on the disc are numbered from 0 to 10, page 0 being reserved for comments, text is then input to the disc by means of the editor described after. To compile a program from RAM-disc, use

n LOAD (where n is the page number of the first definition)

if the definition or definitions, spread over more than one page, the final word on the page should be

--> (pronounced next-screen)

TO prepare the RAM disc for writing, the command

INIT-DISC

should be used. This simply clears the area with blanks. To list the current contents of the disc pages, use

n LIST (where n is the page to be listed)

When a disc has been filled, it can be saved to tape with the command

```
  SAVET
```

it can also be verified subsequently by using

```
  VERIFY
```

If at any time during these operations, an error occurs, sending the Spectrum back into BASIC, you can return to the FORTH system where you left it by typing GOTO 3 (this performs a Warm start).
To reload a disc area created previously use

```
  LOADT
```

but remember, this overwrites whatever is already there.

## 2.8 The Graphics Routines

These routines are the major extensions to the FORTH standard, and allow use of all the Spectrum's Hi-Res graphics routines, which with the speed of Spectrum FORTH allow fast-action arcade-type games to be written without the need to resort to machine code.

```
    n1 n2 SCREEN
    n1 n2 AT
    n1 n2 ATTR
```

(where n1 = line number and n2 = column)

SCREEN returns the ASCII code of the chacter at screen location n1, n2.
AT positions the cursor at n1, n2.
ATTR returns the attributes of the screen at n1, n2 in standard Spectrum format.

```
    n INK
    n PAPER
```

(where n equals a Spectrum colour value from 0 — 9)

```
    n BORDER
```

(where n equals a Spectrum colour value from 0 — 7)
These commands are directly equivalent to three Spectrum cousins.

```
    x y PLOT
    x y DRAW
```

(where x and y are Spectrum screen co-ordinates x = 0 — 255, y = 0 — 175)

Unlike the Spectrum command PLOT the FORTH PLOT simply ignores values that are off the screen and DRAW uses absolute values. So the BASIC commands

```
    PLOT(100, 100)
    DRAW(-20, 20)
```

would in FORTH be

```
100 100 PLOT
80 120 DRAW
```

If you DRAW off the screen, the next DRAW will DRAW from the last visible PLOT on screen.

A useful definition that you can add to your own system is one that will draw a line from one position to another. This can be defined if FORTH as:-

```
: DRAWL PLOT DRAW ;
```

and has a definition of:-

```
x2  y2  x1  y1  ---
```

Notice with this that the TO position comes before the FROM position.

An interesting comparision of the speed of FORTH compared with BASIC can be seen by this trivial program that fills the screen by plotting each dot in turn. In FORTH define a new word TEST:-

```
: TEST 256 0 DO
176 0 DO
J I PLOT
LOOP LOOP
;
```

to run type TEST.
In BASIC, the line:-

```
FOR J = 0 TO 255 : FOR I = 0 TO 175 : PLOT J, I : NEXT I : NEXT J
```

Quite a difference!
(Note also that the final value in a FORTH DO statements is not used.)

The following program uses PLOT and DRAW to draw a pattern on the screen:-

```
: TEST1
251 0 DO
125 0 PLOT 1 88 DRAW
125 175 PLOT 1 88 DRAW
10 +LOOP
;
```

This definition can also be used to show the extensibility of FORTH using previously defined words. If it is decided that what is wanted is the pattern to be repeated in all colours, a new word can be defined using TEST1:-

```
: TEST2
7 0 DO
I INK CLS TEST1
10000 0 DO LOOP
LOOP
;
```

x y POINT
as in the standard Spectrum routines, this command returns either a true or false flag, depending whether the point is set or not.

```
    f FLASH
    f BRIGHT
    f GOVER
    f INVERSE
```

These commands are again, directly related to the Spectrum commands, with a false (0) value turning them off, and a true (<> 0) value turing them on. (The reason for GOVER rather than OVER for overprint is because of the clash with the standard FORTH name OVER, which does something entirely different.)

## 2.9 Sounding Out

There is only the one sound command in Spectrum FORTH and that works in a different way from the BASIC command BEEP, hence the new name BLEEP!
Used in the format:-

```
    n1 n2 BLEEP
```

this produces a tone of duration n1 and pitch n2. These numbers are directly related to machine code cycles and as such are both more difficult to use successfully, and also capable of producing the sounds only heard until now in the commercial software that is available.

## 2.10 The Outside World

Two commands are available for communication through the standard I/O ports of the Spectrum. These are:-

```
    n1 INP
    n1 n2 OUTP
```

INP returns a value to the stack from port n1 and will be useful when writing programs that require the interpretation of more than one key at a time. OUTP put the value n1 out onto port n2.

## 2.11 Miscellaneous Additions

The final changes to the standard are:-

```
    FREE
```

this returns a value on the stack of the amount of store remaining in bytes. For example:-

```
: BYTES   FREE   . . " bytes   remaining"   CR   ;
```

when BYTES is typed the message : 18919 bytes remaining : or similar will be printed.

```
SIZE
```

this returns a value on the stack of the current size of the dictionary.

```
f LINK
```

this command (where f is a flag 0 — false, <>0 — true) links the printer and the screen together so that anything output to the screen will be also printed on the ZXPrinter. This is one of the occasions when using the standard BREAK key may cause problems! ALWAYS use Caps Shift 1 to break a program.

```
UDG
```

UDG returns the position in store of the user-defined graphics. Graphics-A is in UDG+0 to UDG+7, Graphics-B is in UDG+8 to UDG+15 etc. A useful command to define the graphics can be compiled as follows:

```
: DEFINE (take 8 numbers followed by a character number 0 — 21 and use to
  redefine a character *)
  8   *   UDG   +   DUP   8   +   SWAP
  (take top value on stack and use as index for DO LOOP *)
  DO I C! LOOP
  (Store the next 8 numbers from the stack into the user variable)
  ; (finish definition)
```

this command can then be used as follows:-

To define Graphic-C as a hollow square:-

```
HEX
FF 81 81 81 81 81 81 FF 2 DEFINE
```

# CHAPTER 3. ADVANCED FEATURES OF FORTH

## 3.0 Saving an extended Dictionary

As it can be seen from any work of FORTH, a completed FORTH application is simply an extension of the dictionary. In Spectrum FORTH, if you have a completed program and do not wish to have to compile from RAM-disc each time you want to use it, or you have a set of standard routines that you wish to use every time you enter FORTH, these may be saved by the following method:-

1) Find out the new total length of your program by typing:-

```
SIZE.
```

2) Change the COLD START parameters by typing the following lines:-

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE FENCE !
' FORTH 8 + 32 +ORIGIN !
(If using a different vocabulary than FORTH use 'vocab 6 + ...)
```

3) Type:-

```
MON
```

to return to BASIC.

4) Edit line 9 of the BASIC loader program to have a final code length value of the number given in 1) + 10.

5) RUN 9 to save your new extended version of FORTH to tape. (Use your own tape to do this on, **NOT** the master FORTH tape.) This new tape is of course for your own use only and not for re-sale hire or lending purposes.

**3.1 Register Usage.**

The programmer who wishes to use machine code routines (using CREATE or ;CODE) will require the register usage of SPECRUM FORTH. These are as follows:-

| **REGISTERS** | | |
|---|---|---|
| **FORTH** | **Z80** | |
| IP | BC | Must be preserved across words. |
| W | DE | Input only when PUSHDE is used. |
| SP | SP | Data Stack. |
| UP | IX | User area pointer. Must be preserved across FORTH words. |
| | HL | Input only when PUSHHL used. |

To understand this area fully will require Ting's book as mentioned earlier.

## 3.2 Future Plans.

In the near future, we will be bringing out additions to make FORTH a lot more powerful. These include a full Z80 assembler, Micro-Drive file support Floating-point number capacity (although fixed-point is faster and can be used for most purposes) and later still a full cross-compiler that will allow any of your developed software to be compiled into a form that can then be sold as your own program with no restraints or licensing problems with ourselves. All these enhancements (with the exception of the cross-compiler) will be available at a very reasonable cost from ourselves and if you will fill in the registration form at the back of this manual and send it to us, you will be sure of receiving prompt knowledge of these products when they appear.

# CHAPTER 4. THE DISC EDITOR

## 4.0 Arrangement of the RAM-disc.

FORTH organises all mass storage (either RAM-disc or MICRO-DRIVE) as screens of 1024 characters. The RAM-disc has 11K and its screens are numbered 0 to 11.

Each screen is organised by the system into 16 lines of 64 characters per line. The FORTH screens are an arrangement of virtual memory and do not correspond with the Spectrum screen format.

## 4.1 Inputting to a screen.

To start an editing session, type EDITOR. This invokes the EDITOR vocabulary and allows text to be input.

The screen to be edited is selected, using either:-

```
n LIST (list screen n and select for editing) OR
n CLEAR (clear screen n and select for editing)
```

To input text after LIST or CLEAR, the P (put) command is used.

Example:

```
0 P This is how
1 P to input text
2 P onto lines 0, 1, and 2 of the selected screen.
```

## 4.2 Line Editing

During this description of the editor, reference is made to PAD. This is the text buffer. It may hold a line of text used by or saved with a line editing command, or a text string to be found or deleted by a string editing command.

The PAD can be used to transfer a line from one screen to another, as well as to perform edit operations within a single screen.

| | |
|---|---|
| N | Find the next occurrence of the string found by an F command. |
| X text | Find and delete the string "text". |
| C text | Copy in text to the cursor line at the cursor position. |
| TILL text | Delete on the cursor line from the cursor till the end of the text string "text". |
| NOTE: | Typing C with **no text** will copy a null into the text at the cursor position. This will abruptly stop later compiling! To delete this error type TOP X enter. |

# CHAPTER 5. ERROR MESSAGES

If the compiler finds an error at any point, it clears both the data and return stack, and gives an error message with a numeric value. The meaning of these error messages is:-

| MSG# | **Meaning** |
|---|---|
| 0 | Word not found. |
| 1 | Stack empty. |
| 2 | Dictionary full. |
| 3 | Has incorrect address mode. |
| 4 | Is not unique. |
| 6 | RAM Disc Range? (not pages 0 to 10) |
| 7 | Full stack. |
| 9 | Trying to load from page 0. |
| 17 | Compilation only, use in a definition. |
| 18 | Execution only. |
| 19 | Conditionals not paired. |
| 20 | Definition not finished. |
| 21 | In protected dictionary. |
| 22 | Use only when loading. |
| 23 | Off current editing screen. |
| 24 | Declare vocabulary. |

# CHAPTER 6. FORTH & EDITOR GLOSSARIES

The glossary contains all of the word definitions in this release of fig-FORTH (with extensions for the Spectrum). The definitions are presented in the order of their ASCII sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes '---' indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

| | |
|---|---|
| addr | memory address |
| b | 8 bit byte (i.e. high 8 bits zero) |
| c | 7 bit ASCII character |
| d | 32 bit signed double integer. |
| f | boolean flag. 0 = false, non-zero = true. |
| ff | boolean false flag = 0 |
| n | 16 bit signed integer number. |
| u | 16 bit unsigned integer number |
| tf | boolean true flag = non-zero |

The capital letters on the right show definition characters:

| | |
|---|---|
| C | May only be used within a colon definition. A digit indicates number of memory addresses used. |
| E | Intended for execution only. |
| P | Has precedence bit set. Will execute even when compiling. |
| U | A user variable. |

Unless otherwise noted, all references to numbers are for 16 bit signed integers. The high byte is on top of the stack, with the sign in the leftmost bit. For 32 bit numbers the most significant part is on top.

All arithmetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

Acknowledgements are duly made to the FORTH INTEREST GROUP. PO Box 1105, San Carlos, CA 94070 for parts of this compiler and manual.

## Line Editor Commands

| | |
|---|---|
| n H | Hold line n at PAD (used by system more often than by user) |
| n D | Delete line n but hold it in PAD. Line 15 becomes blank as lines n+1 to 15 move up 1 line. |
| n T | Type line n and save it in PAD. |
| n I | Insert the text from pad at line n, moving the old line n and following lines down. Line 15 is lost. |
| n E | Erase line n with blanks. |
| n S | Spread at line n. n and subsequent lines move down 1 line. Line n becomes blank. Line 15 is lost. |

## 4.3 String Editing.

The screen of text being edited resides in a buffer area of storage. The editing cursor is a variable holding an offset into this buffer area. Commands are provided for the user to position the cursor, either directly or by searching for a string of buffer text, and to insert or delete text at the cursor position.

### Commands to Position the Cursor

| | |
|---|---|
| TOP | Position the cursor at the start of the screen. |
| n M | Move the cursor by a signed amount n and print the cursor underscore. |
| n LIST | List screen n and select it for editing. |
| n CLEAR | Clear screen n with blanks and select it for editing. |
| n1 n2 COPY | Copy screen n1 to screen n2. |
| L | List the current screen. The cursor line is relisted after the screen listing, to show the cursor position. |
| F text | Search forward from the current cursor position until string "text" is found. The cursor is left at the end of the text string, and the cursor line is printed. If the string is not found an error message is given and the cursor is repositioned at the top of screen. |
| B | Used after F to back up the cursor by the length of the most recent text. |

!　　n addr ---　　LO

Store 16 bits of n at address. Pronounced "store"

!CSP

Save the stack position in CSP.Used as part of the compiler security.

#　　d1 --- d2　　LO

Generate from a double number d1, the next ASCII character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.

#>　　d --- addr count　　LO

Terminates numeric output conversions by dropping d, leaving the text address and character count suitable for TYPE.

#BUF --- n

A constant returning the number of disc buffers allocated.

#S　　d1 --- d2　　LO

Generates ASCII text in the text output buffer, by the use of #, until a zero double number results. Used between <# and #>.

'　　--- addr　　P,LO

Used in the form:

' nnnn

Leaves the parameter field address of dictionary word nnnn. As a compiler directive, ' executes a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".

(　　P,LO

Used in the form:

( cccc)

Ignore a comment that will be delimited by a right paranthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.

(.")　　C+

The run-time procedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

(;CODE)　　C

The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP) n ---　　C2

The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

(ABORT)

Executes after an error when WARNING is -1. The word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.

(DO)　　C

The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO

(FIND) addr1 addr2 --- pfa b tf　　(ok)
addr1 addr2 --- ff　　(bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and true for a good match. If no match is found, only a boolean false is left.

(LINE) n1 n2 --- addr count

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP) C2
The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP

(NUMBER) d1 addr1 --- d2 addr2
Convert the ASCII text beginning at addr1 + 1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

(TAPE) n ---
The primitive used by all the tape routines.

* n1 n2 --- prod LO
Leave the signed product of two signed numbers.

*/ n1 n2 n3 --- n4 LO
Leaves the ratio n4=n1*n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence:
n1 n2 * n3 /

*/MOD n1 n2 3n --- n4 n5 LO
Leave the quotient n5 and remainder n4 of their operation n1*n2/n3. A 31 bit intermediate product is used as for */

+ n1 n2 --- sum LO
Leave the sum of n1 + n2.

+! n addr --- LO
Add n to the value at the address. Pronounced "plus-store".

+− n1 n2 --- n3
Apply the sign of n2 to n1, which is left as n3.

+BUF addr1 --- addr2 f
Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP n1 --- (run)
addr n2 --- (compile) P,C2,LO
Used in a colon-definition in the form:
DO ... n1 +LOOP

At run-time, +LOOP selectively controls branching back to the corresponding DO, based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

+ORIGIN n --- addr
Leave the memory address offset n bytes from the origin. This definition is used to access or modify the boot-up parameters at the origin area.

.CPU
Prints the message '48K Spectrum'.

, n --- LO
Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

− n1 n2 --- diff LO
Leave the difference of n1 − n2.

--> P,LO
Continue interpretation with the next disc screen. (pronounced next-screen).

−DUP n1 --- n1 (if zero)
n1 --- n1 n1 (if non-zero) LO
reproduce n1 only if it is non-zero.

This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

–FIND ---pfa b tf (found)
--- ff (not found)
Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

–TRAILING addr n1 --- addr n2
Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks.

. n --- LO
Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blank follows. Pronounced "dot".

." P,L
Used in the form:
." cccc"
Compiles an in-line string cccc (delimited by the trailing ") with an execution procedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". See (.").

.LINE line scr ---
Print on the screen, a line of text from the RAM disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---
Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 --- rem
Leave the signed quotient of n1/n2.

/MOD n1 n2 --- rem quot LO
Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n
These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.

0< n --- f LO
Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= n --- f LO
Leave a true flag if the number is equal to zero, otherwise leave a false flag.

)BRANCH f ---
The run-time procedure to conditionally branch. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL and WHILE.

1+ n1 --- n2 L1
Increment n1 by 1.

2+ n1 --- n2
Increment n1 by 2.

2! nlow nhigh addr ---
32 bit store, nhigh is stored at addr; nlow is stored at addr+2.

2CONSTANT d ---
A defining word used in the form:
d 2CONSTANT cccc
to create word cccc, with its parameter field containing d. When cccc is later executed, it will push the double value of d to the stack.

2DROP d ---
Drop the double number from the stack.

2DUP n2 n1 --- n2 n1 n2 n1
Duplicates the top two values on the stack. Equivalent to OVER OVER.

2OVER d1 d2 --- d1 d2 d1
Copy the first double value d1 to the top of the stack.

2SWAP d1 d2 --- d2 d1
Exchange the top two double numbers on the stack.

2VARIABLE
A defining word used in the form:
d VARIABLE ccc
When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to d. When cccc is later executed, the address of its parameter field (containing d) is left on the stack, so that a double fetch store may access this location.

: P,E,LO
Used in the form called a colon-definition:
: cccc ... ;
Creates a dictionary entry defining cccc as equivalent to the following sequence of FORtH word definitions '...' until the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

; P,C,LO
Terminate a colon-definition and stop further compilation. Compiles the run-time;S.

;CODE P,C,LO
Used in the form:
: cccc .... ;CODE
Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics. If the ASSEMBLER is not loaded, code values may be compiled using , and C,.When cccc later executes in the form:
cccc nnnn
the word nnnn wil be created with its execution procedure given by the machine code following cccc. That is when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P,LO
Stop interpretation of a screen. ;S is also the run-time word compiled at he end of a colon-definition which returns execution to the calling procedure.

< n1 n2 --- f LO
Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<# LO
Setup for pictured numeric output formatting using the words:
<# # #S SIGN #>
The conversion is done on a double number producing text at PAD.

<BUILDS C,LO
Used within a colon-definition:
: cccc <BUILDS...
DOES> ... ;
Each time cccc is executed, <BUILDS defines a new word with a high-level execution procedure. Executing cccc in the form:
cccc nnnn
uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time procedures to be written in high-level rather than in assmebler code (as required by ;CODE).

= n1 n2 --- # LO
Leave a true flag if n1 = n2; otherwise leave a false flag.

< n1 n2 --- # LO
Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n --- C,LO
Remove a number from the computation stack and place as the most accessable on the return stack. Use should be balanced with R> in the same definition.

? addr --- LO
Print the value contained at the address in free format according to the current base.

?COMP
Issue error message if not compiling.

?CSP
Issue error message if stack position differs from value saved in CSP.

?ERROR f n ---
Issue an error message number n, if the boolean flag is true.

?EXEC
Issue an error message if not executing.

?LOADING
Issue an error message if not loading.

?PAIRS n1 n2 ---
Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

?STACK
Issue an error message if the stack is out of bounds.

?TERMINAL--- f
Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation.

@ addr --- n LO
Leave the 16 bit contents of address.

ABORT LO
Clear the stacks and enter the execution state. Return control to the keyboard, printing a message.

ABS n --- u LO
Leave the absolute value of n as u.

AGAIN addr n --- (compiling)
P,C2,LO
Used in a colon-definition in the form:
BEGIN ... AGAIN
At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below). At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n --- LO
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or reorigin memory. n is with regard to computer address type (byte or word).

AND n1 n2 --- n2 LO
Leave the bitwise logical and of n1 and n2 as n3.

AT n1 n2 ---
Moves the cursor position to line n1, column n2.

ATTR n1 n2 --- b
Returns the attribute byte at line n1, column n2.

B/BUF --- n
This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK

B/SCR --- n

This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organised as 16 lines of 64 characters each.

BACK addr ---

Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE --- addr U,LO

A user variable containing the current number base used for input and output conversion.

BEGIN --- addr n (compiling P,LO

Occurs in a colon-definition in form:
BEGIN . . . UNTIL
BEGIN . . . AGAIN
BEGIN . . . WHILE . . . REPEAT

At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

At compile time BEGIN LEAVES its return address and n for compiler error checking.

BL --- c

A constant that leaves the ASCII values for "blank".

BLANKS addr count ---

Fill an area of memory beginning at addr with count blanks.

BLEEP n1 n2 ---

Produce a tone in the Spectrum noise maker of duration n1, pitch n2.

BLK --- addr U,LO

A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK n --- addr LO

Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from microdrive to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to microdrive before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH

BRANCH C2,LO

The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.

BRIGHT f ---

Set the screen to bright if flag f is true, else unset bright.

BORDER n ---

Set the border to colour n.

BUFFER n --- addr

Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the microdrive. The block is not read from the microdrive. The address left is the first cell within the buffer to data storage.

C! b addr ---

Store 8 bits at address.

C/L --- n

Constant leaving the number of characters per line; used by the editor.

C, b ---

Store 8 bits of b into the next available dictionary byte, advancing the dictonary pointer.

C@ addr --- b

Leave the 8 bit contents of memory address.

CASE --- n (compiling)

Occurs in a colon definition in the form:

CASE

n OF . . . . . ENDOF

. . . . .

ENDCASE

At run-time, CASE marks the start of a sequence of OF . . . ENDOF statements.

At compile-time CASE leaves n for compiler error checking.

CFA pfa --- cfa

Convert the parameter field address of a definition to its code field address.

CLS ---

Performs the clear-screen home-cursor function.

CMOVE from to count ---

Move the specified quantity of bytes beginning at address from , to address to. The contents of address from is moved first , proceeding toward high memory.

COLD

The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

COMPILE C2

When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictonary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).

CONSTANT n --- LO

A defining word used in the form:

n CONSTANT cccc

to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT --- addr U,LO

A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT adr1 --- addr2 n LO

Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

CR LO

Transmit a carriage return and line feed to the selected output device.

CREATE

A defining word used in the form:

CREATE cccc

by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the word's parameter field. The new word is created in the current vocabulary.

CSP --- addr U

A user variable temporarily storing the stack pointer position, for compilation error checking.

D+ d1 d2 --- dsum

Leave the double number sum of two double numbers.

D+− d1 n --- d2

Apply the sign of n to the double number d1, leaving it as d2.

D. d --- L1

Print a signed double number from

a 32 bit two's complement value. The high-order 16 bits are most accessable on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R. d n ---
Print a signed double number d, right aligned in a field n characters wide.

DABS d --- ud
Leave absolute value ud of a double number.

DECIMAL LO
Set the numeric conversion base for decimal input-output.

DEFINITIONS LI
Used in the form:
cccc DEFINITIONS
Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made if both the CONTEXT and CURRENT vocabularies.

DIGIT c n1 --- n1 tf (ok)
c n1 --- ff (bad)
Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLITERAL d --- d (executing)
d --- (compiling) P
If compiling, compile a double number from the stack into a literal. Later execution of the defintition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS d1 --- d2
Convert d1 to its double number two's complement.

DO n1 n2 --- (execute)
addr n --- (compile)
P,C2,LO
Occurs in a colon-defintition in form:
DO . . . LOOP
DO . . . +LOOP
At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one, and B, unless A. In this case the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling with the colon-definition, DO compiles (DO). It then leaves the following address addr, and n for later error checking.

DOES> LO
A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multidimensional arrays, and compiler generation.

DP --- addr U,L
A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.

DPL --- addr U,LO
A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used to hold output column location of a decimal point, in user generated formating. The default value on single number input is – 1.

DRAW n1 n2 ---
Draws a line from the current plot position to n1(x),n2(y). Any points drawn off the screen are ignored.

DROP n --- LO
Drop the number from the stack.

DUP n --- n n LO
Duplicate the value on the stack.

EDITOR
The editor vocabulary.

ELSE addr1 n1 --- addr2 n2
(compiling) P,C2,LO
Occurs within a colon-definition in the form:
IF . . . ELSE . . . ENDIF
At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE compiles BRANCH, reserving space for a branch offset. It leaves the address addr2, and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE, and storing at addr1.

EMIT c --- LO
Transmit ascii character c to the selected output device. OUT is incremented for each character output.

EMPTY-BUFFERS LO
Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not writen to the microdrive. This is also an initializtion procedure before first use of the microdrive.

ENCLOSE addr1 c ---
addr1 n1 n2 n3
The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ascii 'null", treating it as an unconditional delimiter.

END P,C2,LO
This is an 'alias' or duplicate definition for UNTIL.

ENDCASE addr n --- (compile)
Occurs in a colon definition in the form:
CASE
n OF . . . . . ENDOF
. . . . .
ENDCASE
At run-time ENDCASE marks the conclusion of a CASE statement.

At compile-time, ENDCASE computes forward branch offsets.

ENDIF addr n --- (compile) P,CO,LO
Occurs in a colon-definition in form:
IF . . . ENDIF
IF . . . ELSE . . . ENDIF
At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. it marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

ENDOF Addr n --- (compile)
Used as ENDIF but in CASE statements.

ERASE addr n ---
Clear to zero a region of memory, n bytes long, starting at addr.

ERROR line --- in blk
Execute error notification and restart of systems. WARNING is first examined. If 1, the text of line n, relative to screen 4 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n is just printed as a message number (RAM disc installation). If WARNING is −1 the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK on the stack to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr ---
Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXIT
Force the conclusion of a definition at a different place than the end. Must not be used when the return stack has been modified!

EXPECT addr count --- LO
Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.

FENCE --- addr U
A user variable containing an address below which FORGETting is not allowed. To forget below this point the user must alter the contents of FENCE.

FILL addr quan b ---
Fill memory at the address with the specified quantity of bytes b.

FIRST --- n
A constant that leaves the address of the first (lowest) block buffer.

FLASH f ---
Sets screen to flash if f true else sets no-flash.

FLD --- addr U
A user variable for control of number output field width. Presently unused in fig-FORTH.

FLUSH
Write all updated disc buffers to the microdrive.

FORGET E,LO
Executed in the form:
FORGET cccc
Deletes definition named cccc from the dictionary with all entries physically following it. In fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH P,L1
The name of the primary vocabulary. Execution makes FORTH the CONTEXT VOCABULARY. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition to select this vocabulary at compile time.

HERE --- addr LO
Leave the address of the next available dictionary location.

HEX LO
Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr LO
A user variable that holds the address of the latest character of

text during numeric output conversion.

HOLD c --- LO
Used between <# and #> to insert an ascii character into a pictured numeric output string.
e.g. 2E HOLD will place a decimal point.

I --- n C,LO
Used within a DO-LOOP to copy the loop index to the stack.
See R.

I' --- n
Copies the last but one value from the return stack (which within a DO-LOOP is the loop limit).

ID. addr ---
Print a definition's name from its name field address.

IF f --- (run-time)
--- addr n (compile) P,C2,LO
Occurs in a colon-definition in the form:
IF (tp) . . . ENDIF
IF (tp) . . . ELSE (fp)
. . . ENDIF
At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead throughout the true part. If f is false, execution jumps to just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional. if missing, false execution skips to just after ENDIF.
At compile-time IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IMMEDIATE
Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled, i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceeding it with [COMPILE].

IN --- addr LO
A user variable containing the byte offset within the current input text buffer (terminal or RAM-disc) from which the next text will be accepted. WORD uses and moves the value of IN.

INDEX from to ---
Print the first line of each screen over the range from, to. This is used to view the comment lines of an area of text on RAM-disc screens.

INIT-DISC
Wipes all information off the RAM disc prior to first use.

INK n ---
Sets the ink colour to n(0-9)

INKEY --- c
Leaves the value of the key being pressed. If no key being pressed leaves hex. FF.

INP n1 --- b
Returns the value read from port n1.

INTERPRET
The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or RAM-disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a "?" will; be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose

than to force this action. See NUMBER.

INVERSE f ---
Sets the screen to inverse if true else sets to normal.

J --- n
Used within nested DO loops. Returns the index value of the outer loop.

KEY --- v LO
Leave the ascii value of the next terminal key struck.

LATEST --- addr
Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE C,LO
Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.

LFA pfa ---lfa
Convert the parameter field address of a dictionary definition to its link field address.

LIMIT --- n
A constant leaving the address just above the highest memory address used by the system.

LINE n --- addr
Leave address of line n of current screen. This address will be in the RAM-disc buffer area.

LINK f ---
Joins the printer to the screen if the flag is true else disconnects it.

LIST n --- LO
Display the ASCII text of screen n on the selected output device. SCR contains the screen number during and after this process.

LIT --- n C2,LO
Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.

LITERAL n --- (compiling) P,C2,LO
If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition, the intended use is:
: xxx [calculate] LITERAL ;
Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles this value.

LOAD n --- LO
Begin interpretation of screen n. Loading will terminate at the end of the screen or at;S. See ;S and -->.

LOADT
Loads the information of a RAM disc area previously saved with SAVET.

LOOP addr n --- (compiling) P,C2,LO
Occurs in a colon-definition in form:
DO . . . LOOP
At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.

At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.

M*   n1 n2 --- d
A mixed magnitude math operation which leaves the double number signed product of two signed numbers.

M/   d n1 --- n2 n3
A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.

M/MOD   ud1 u2 --- u3 ud4
An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.

MAX   n1 n2 --- max   LO
Leave the greater of two numbers.

MESSAGE   n ---
Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (RAM disc system).

MIN   n1 n2 --- min   LO
Leave the smaller of two numbers.

MINUS   n1 --- n2   LO
Leave the two's complement of a number.

MOD   n1 n2 --- mod   LO
Leave the remainder of n1/n2, with the same sign as n1.

MON
Exit to Basic. Error 9 (stop).

NEXT
This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code procedures. It acts by fetching the address pointed to by IP, storing this value in register W. W points to the code field of a definition which contains the address of the code which is to be executed for that definition. This address is then jumped to. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific. (See earlier note on Sectrum FORTH convention).

NFA   pfa --- nfa
Convert the parameter field address of a definition to its name field.

NOOP
A FORTH no-operation.

NOT   f --- f
Leaves a false flag if a true flag is on the stack, else leaves a true flag. (Actually executes 0=)

NUMBER   addr --- d
Convert a character string left at addr with a preceeding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given.

OFFSET   --- addr   U
A user variable which may contain a block offset for the RAM-disc. The contents of OFFSET is added to the stack number by BLOCK. Messages printed by MESSAGE are independent of OFFSET. See BLOCK, MESSAGE.

OR   n1 n2 --- or   LO
Leave the bit-value logical or of two 16 bit values.

OUT   --- addr   U
A user variable that contains a value incremented by EMIT. The user

may alter and examine OUT to control display formating.

OUTP n1 n2 ---
Presents n1 to output port n2.

OVER n1 n2 --- n1 n2 n1 LO
Copy the second stack value, placing it as the new top.

PAD --- addr LO
Leaves the address of the text output buffer, which is a fixed offset above HERE.

PAPER n1 ---
Sets the paper colour to n1(0-9)

PFA nfa --- pfa
Convert the name field address of a compiled definition to its parameter field address.

PLOT n1 n2 ---
Sets point n1(x),n2(y). If this value is off the screen, no action is taken.

POINT n1 n2 --- f
Returns a true flag if n1(x),n2(y) is set, else returns a false flag.

PREV --- addr
A variable containing the address of the buffer most recently referenced. The UPDATE command marks this buffer to be later written to microdrive.

PUSHDE
PUSHHL
This code sequence pushes machine registers to the computation stack and returns to NEXT. it is not directly executable, but is a Forth re-entry after machine code. PUSHHL returns just the HL register PUSHDE returns the DE register under the HL register on the stack.

QUERY
Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT
Clear the return stack, stop compilation, and return control to the operator's terminal. No message is given.

R --- n
Copy the top of the return stack to the computation stack.

R# --- addr U
A user variable which may contain the location of an editing cursor or other file related function.

R/W addr blk f ---
The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer. blk is the sequential number of the referenced block; and f is a flag for f= write and f=1 read. R/W determines the location on the microdrive, performs the read-write and performs any error checking.

R> --- n LO
Remove the top value from the return stack and leave it on the computation stack. See >R and R.

R0 --- addr U
A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!

REPEAT addr n --- (compiling) P,C2
Used within a colon-definition in the form:
BEGIN . . . WHILE . . . REPEAT
At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 --- n2 n3 n1 LO
Rotate the top three values on the stack, bringing the third to the top.

RP@ addr
Leaves the current value in the return stack pointer register.

RP!
A computer dependent procedure to initialize the return stack pointer from user variable R0.

S->D n --- d
Sign extend a single number to form a double number.

S0 --- addr U
A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SAVET
Save a RAM disk area to tape with the name DISC. Can be re-loaded with LOADT.

SCR --- addr
A user variable containing the screen number most recently referenced by LIST.

SCREEN n1 n2 --- a
Returns the ascii value of the character at line n1, column n2 as long as that character is not a user-defined character.

SIGN n d --- d LO
Stores an ascii "–" sign just before a converted numeric output string in the next output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.

SMUDGE
Used during word definition to toggle the "smudge bit" in a definitions' name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.

SP!
A computer dependent procedure to initialize the stack pointer from SO.

SP@ --- addr
A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)

SPACE LO
Transmit an ascii blank to the output device.

SPACES n --- LO
Transmit n ascii blanks to the output devic.

STATE --- addr LO,U
A user variable containing the compilation state. A non-zero value indicates compilation.

SWAP n1 n2 --- n2 n1 LO
Exchange the top two values on the stack.

TASK
A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

TEXT c ---
Accept following text to PAD. c is the text delimiter.

THEN P,CO,LO
An alias for ENDIF.

TIB --- addr U
A user variable containing the address of the terminal input buffer.

TOGGLE addr b ---
Complement the contents of addr by the bit pattern b.

TRAVERSE addr1 n --- addr2
Move across the name field of a fig-FORTH, variable length, name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=-1, the motion is toward low memory. The addr resulting is address of the other end of the name.

TRIAD scr ---
Display on the selected output device the three screens which include the one numbered scr, beginning with a screen evenly divisible by three. Output is suitable for source text records.

TYPE addr count --- LO
Transmit count characters from addr to the selected output device.

U< u1 u2 --- f
Leave the boolean value of an unsigned less-than comparison. Leaves f = 1 for u1 < u2; otherwise leaves 0. This function should be used when comparing memory addresses.

U* u1 u2 --- ud
Leave the unsigned double number product of two unsigned numbers.

U. u ---
Prints an unsigned 16-bit number converted according to BASE. A trailing blank follows.

U.R u n ---
Print the unsigned number u right aligned in a field whose width is n. No following blank is printed.

U/MOD ud u1 --- u2 u3
Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UNTIL f --- (run-time)
addr n --- (compile) P,C2,LO
Occurs within a colon-definition in the form:
BEGIN . . . UNTIL
At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.
At compile-time, UNTIL compiles (0BRANCH) and an offset from HERE to addr. n is used for error tests.

UDG
UDG returns the position in store of the user-defined graphics. Graphics-A is in UDG + 0 to UDG + 7, Graphics-B is in UDG + 8 to UDG + 15, etc

UPDATE LO
marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disk should its buffer be required for storage of a different block.

USE --- addr
A variable containg the address of the block buffer to use next, as the least recently written.

USER n --- LO
A defining word used in the form:
n USER cccc
The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and user area base address on the stack as the storage address of that particular variable.

VARIABLE E,LU
A defining word used in the form:
n VARIABLE cccc
When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VERIFY
Used to check a tape created by SAVET. If an error is found, the system reverts to BASIC. The FORTH can be re-entered without

losing any work by typing GOTO 3. SAVET can then be tried again.

VLIST

List the names of the definitions in the context vocabulary. "Break" will terminate the listing.

VOC-LINK --- addr U

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control by FORGETting through multiple vocabularies.

VOCABULARY E,L

A defining word used in the form:

VOCABULARY cccc

to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.

In fig-FORTH, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularies ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

WARNING --- addr U

A user variable containing a value controlling messages. If it = 1 disk is present, and screen 4 of drive 0 is the base location for messages. If it = 0, no disk is present and messages will be presented by number. If it = −1, execute (ABORT) for user defined procedure. See MESSAGE, ERROR.

WHERE n1 n2 ---

If an error occurs during LOAD from disk, ERROR leaves these values on the stack. WHERE uses these values to show the user where the error occurred by printing the screen and line no.

WHILE f --- (run-time)

ad1 n1 --- ad1 n1

ad2 n2 P,C2

Occurs in a colon-definition in the form:

BEGIN . . . WHILE (tp) . . . REPEAT

At run-time, WHILE selects conditional execution based on boolean flag f.

If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE compiles (BRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr U

In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definitions' name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD c --- LO

Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, then the characters, and ends with two or more blanks. Leading occurrences of c are ignored. If BLK is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in BLK. See BLK, IN.

X

This is pseudonym for the "null" or dictionary entry for a name of one character of ascii null. It is the execution procedure to terminate interpretation of a line of text from the terminal or within a disc buffer, as both buffers always have a null at the end.

XOR n1 n2 --- x0r −L1

Leave the bitwise logical exclusive — or of two values.

[ P,L1

Used in a colon-definition in form:

: xxx [ words ] more ;

Suspend compilations. The words after [ are executed, not compiled. This allows calculations of compilation exceptions before resuming compilation with ]. See LITERAL, ].

[COMPILE] P,C

Used in a colon-definition in form:

: xxx [COMPILE] FORTH ;

[COMPILE] will force the compilation of an immediate definition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executed, rather than at compile time.

] L1

Resume compilation, to the completion of a colon-definition. See [.

## EDITOR GLOSSARY

MATCH add1 n1 add2 n2 ---
tf n3

#LOCATE n1 n2
From the cursor position, determine the line-no n2 and the offset into line n1.

#LEAD --- line-address offset-to-cursor

#LAG --- cursor-address count-after-cursor-till-EOL

—MOVE addr line-no ---
Move a line of text from addr to line of current screen.

H n ---
Hold numbered line at PAD.

E n ---
Erase line n with blanks

S n ---
Spread. Lines n and following move down. n becomes blank.

D n ---
Delete line n, but hold in PAD.

M n ---
Move cursor by a signed amount and print its line.

T n ---
Type line n and save in PAD.

L ---
List the current screen.

R n ---
Replace line n with the text in PAD.

P n ---
Put the following text on line n.

I n
Spread at line n and insert text from PAD.

TOP ---
Position editing cursor at top of screen.

CLEAR n ---
Clear screen n, can be used to select screen n for editing.

COPY n1 n2 ---
Copy screen n1 to screen n2.

—TEXT Add1 count Add2 --- f
True if strings exactly match.
add1 n1 add2 n2 --- ff n4
Match the string at add2 with all strings on the cursor line forward from the cursor. If found leaves n3 bytes until the end of the matching string, else leaves n4 bytes to EOL.

1LINE --- f
Scan the cursor line for a match to PAD text. Return flag and update R# to the end of matching text, or to the start of the next line if no match found.

FIND ---
Search for a match to the string at PAD, from the cursor position until the end of screen. If no match found issue an error message and reposition the cursor at the top of the screen.

DELETE n ---
Delete n characters prior to the cursor.

N ---
Find next occurrence of PAD text.

F ---
Input following text to PAD and search for match from cursor position till end of screen.

B ---
Backup cursor by text in PAD.

X ---
Delete next occurrence of following text.

TILL ---
Delete on cursor line from cursor to the end of the following text.

C ---
Spread at cursor and copy the following text into the cursor line.

Now with Abersoft FORTH you can program your Spectrum with a language even more flexible than BASIC, yet almost as powerful as machine code.

Abersoft FORTH is a high level language that will enable you to run programs up to fifty times faster than those written in BASIC, without the tedious programming requirements of machine language.

You can even re-write the language to suit your own programming applications. Abersoft FORTH is *the* tool for programmers who want to squeeze as much power as possible from their Sinclair Spectrum.

"FORTH is an easy language to use, and the graphics commands in Abersoft FORTH allow you to do anything in FORTH that you can do in BASIC. However, the extra speed of FORTH makes a big difference. You can get speed improvements of ten to 50 times, making it possible to produce moving graphics you would otherwise have to write in machine code."

PERSONAL COMPUTER NEWS

"FORTH is more than 10 times faster than BASIC and a game of Space Invaders written using it would move almost as fast as if written in machine code.

Abersoft FORTH is a complete version of the language with added colour, attribute and graphics instructions which can be used to zap space invaders around the screen, if you are that way inclined. It is also possible to define your own characters just as you can in Spectrum BASIC.

It is the only Spectrum package which has been endorsed by the FORTH Interest Group."

SINCLAIR USER

"As far as the Spectrum is concerned then, an implementation of FORTH is going to have to be fast, comprehensive and compact in order to compete with machine code. Abersoft FORTH is — in some very comprehensive tests held recently on four different FORTHs for the Spectrum, Abersoft's came top in every category."

WHICH MICRO

Published by

**MELBOURNE HOUSE**

131 Trafalgar Road, London SE10, United Kingdom
4/75 Palmerston Cres., South Melbourne, 3205, Australia